Reinforcement Learning: An Introduction

Solutions: Chapter 4

Mrinank Sharma

March 31, 2020

Exercise 4.1

$$q_\pi(11, \texttt{down}) = -1 + v_\pi(\texttt{terminal}) = -1. \tag{1}$$
$$q_\pi(7, \texttt{down}) = -1 + v_\pi(11) = -1 - 14 = -15. \tag{2}$$

Exercise 4.2: MDP Modifications

Unchanged dynamics from current states. In this case, the value of these states from these policies is unchanged, we've simply introduced a new state. We can calculate the value of this state using the Bellman equations:

$$v_\pi(15) = -1 + \frac{1}{4}[-22 - 20 - 14 + v_\pi(15)] \Rightarrow v_\pi(15) = -20. \tag{3}$$

Dynamics changed from state 13. In general, we'd expect this to change the value of state 13, which would propagate to all of the other states, meaning that we'd have to recalculate the values. Let's pretend that we were running iterative policy evaluation using the previous value function as our initialisation. Let's update the value for state 13:

$$v_\pi(13) = -1 + \frac{1}{4}[-20 - 22 - 14 \quad \underbrace{-20}_{\text{Estimate for } v_\pi(15)} \quad] = -20 \tag{4}$$

The value of state 13 **hasn't changed!** Therefore, the value of state 15, and all of the other states also won't change. We've recalculated values without having to solve those annoying equations, which is nice.

Exercise 4.3: Iterative Action-Value Evaluation

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r | s, a)[r + \gamma \sum_{a \in \mathcal{A}(s')} \pi(a'|s')[q_\pi(s', a')]], \end{aligned} \tag{5}$$

which can straightforwardly be turned into an iterative update equation.

Exercise 4.4: Policy Iteration Bug

If policy is switching between policies which are equally good, both of the policies are optimal. There must be multiple actions in each step which give the same expected optimal return, and the suggested algorithm does not state how to break ties. We should be able to fix this by settling ties in the arg max operation, for example, by indexing all of the actions and always choosing the lowest index amongst optimal options.

Algorithm 1 Policy Iteration

Initialisation: initialise $V(s)$ arbitarily for all $s \in \mathcal{S}$ except that $V(\text{terminal}) = 0$. Initalise $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$.
Parameters: $\theta > 0$, a threshold determining accuracy.

loop: ▷ Policy Evaluation
  $\Delta \leftarrow 0$
  for each $s \in \mathcal{S}$ **do**:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

`policy-stable` $\leftarrow$ True ▷ Policy Improvement
for each $s \in \mathcal{S}$ **do**:
  `old-action` $\leftarrow \pi(s)$
  $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)[r + \gamma V(s')]$ ▷ Settle ties by index.
  if `old-action` $\neq \pi(s)$ **then**:
    `policy-stable` $\leftarrow$ True
if `policy-stable` **then**:
  return $V \simeq v_*, \pi \simeq \pi*$
else:
  go to 1

Exercise 4.5: Policy Iteration with $q(s, a)$

I will assume deterministic policies here, using the same bug fix.

Algorithm 2 Q Iteration

Initialisation: initialise $Q(a, s)$ arbitarily for all $s \in \mathcal{S}, a \in \mathcal{A}$ except that $Q(\text{terminal}, a) = 0 \quad \forall a$. Initalise $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$.
Parameters: $\theta > 0$, a threshold determining accuracy.

loop: ▷ Policy Evaluation
  $\Delta \leftarrow 0$
  for each $s \in \mathcal{S}$ **do**:
    for each $a \in \mathcal{A}(s)$ **do**
      $q \leftarrow Q(s, a)$
      $Q(s, a) \leftarrow \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)[r + \gamma Q(s', \pi(s'))]$
      $\Delta \leftarrow \max(\Delta, |q - Q(s, a)|)$
until $\Delta < \theta$

`policy-stable` $\leftarrow$ True ▷ Policy Improvement
for each $s \in \mathcal{S}$ **do**:
  `old-action` $\leftarrow \pi(s)$
  $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}(s)} Q(s, a)$ ▷ Settle ties by index.
  if `old-action` $\neq \pi(s)$ **then**:
    `policy-stable` $\leftarrow$ True
if `policy-stable` **then**:
  return $Q \simeq q_*, \pi \simeq \pi*$
else:
  go to 1

Exercise 4.6: ϵ-soft policies

Let's restrict ourself to determinstic + ϵ-soft policies. i.e.,

$$\pi(a|s) = \begin{cases} 1 - \frac{|\mathcal{A}(s)|-1}{|\mathcal{A}(s)|}\epsilon, & a = \pi_d(s) \\ \frac{\epsilon}{\mathcal{A}(s)}, & \text{otherwise} \end{cases}, \tag{6}$$

where π_d is now effectively the determinstic part of the policy. Now, we make the following changes:

1. No change to the initialise of the algorithm.
2. Update the value of the policy by adding a marginalisation step over the possible actions to be selected.
3. In policy improvement, update only the determinstic part of the policy, but using the value of the soft policy to perform the updates.

Note: not fully clear to me that considering this form is policy is sufficient. There are many stochastic policies which we could add a soft part to.

Exercise 4.7: Policy Iteration

Note that we cheated by truncating the poisson distributions (as in the Github replication of the original figure) and also returned a constant number of cars (also as in the reproduction) to make the code run faster. Relevant code is on the Github repo. The replication from the book is found in Fig. 1.

The solution to the modified problem can be seen in Fig. 2. The shape is pretty wacky. As expected, we play 1 quite frequently now because it is free for us to move a car from location 1 to 2 (and it it also often makes sense because of the demands are these places).

State and action space isn't that big here and still dynamic programming was very very slow! We had to perform truncation and an incorrect approximation to get this solution. This shows the limitation of this method, as wonderful as it is.

Exercise 4.8: Gambler's Problem I

I'm a bit confused about this problem! It definitely seems super strange to me. Here are some thoughts:

- The coin is biased *against* us. Paths which require shorter succesful flips are much more likely to happen, and so we should favour them.
- Gambling as much as possible, but not more than needed seems to be good. This always give a (minimum of) 40% of winning. Indeed, from 50 dollars , this is what the optimal strategy does.
- It seems super strange to me that betting 50 dollars is optimal at 50, but if you have one more dollar, you should bet only 1? I would have thought betting 49 dollars in this case would be best.

I looked a bit online and it seems that families of optimal policies are basically the bets which could take us to the peaks of the policy graph e.g., from 51 bets could be $1, 24$ or 49. But honestly, I don't really see why.

Since the coin is against us, if we bet on making lots of conservative bets, in expected value, the value will keep going down and down.

$$\mathbb{E}[c_{t+1}] = c_t - b + p \cdot 2b = c_t - b(1-2p) \tag{7}$$

So in general, we ought to minimise the expected number of flips to success. If we made loads of small bets, basically the law of large numbers would mean that eventually we'd definitely lose all of our money. But, if we make small numbers of bets, we basically are far more likely to *get lucky*, so in general, this is what we should do. Its interesting that this is **not** what the strategy does! It doesn't bet it all when capital is greater than fifty, even though it could.

See. Fig. 3 for a reproduction. This shows that there are indeed multiple optimal moves. The strategy of always betting the most you can works well. Its funny that move of 0 always seems to be optimal too, showing that you can never increase your chance of winning on average. The strategy seems to head to the attractor points.

I think there is a lot more that could be said for this problem.

I don't think that this is particularly insightful. If you have anything better, please contact me!

Exercise 4.9: Programming the Gamblers Problem

Please see Figures 4, 4, 6, 7 for results.

The solution is not stable when the coin is biased to give the player the advantage. Note that the solutions that we do get however play much more conservatively - we want to play many games now, because, on average, we expect to win.

When the coin is even more biased against us, we get the same strategy with less value, which is not surprising.

Note that we haven't employed discounting here (i.e., we don't care how long we are playing for). If we did, I think the solution would be stable (though I haven't tried it).

Exercise 4.10: Q-Value Iteration

$$q_{k+1}(s,a) = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)[r + \gamma \max_{a' \in \mathcal{A}(s')} q_k(s', a')], \tag{8}$$

which is very similar to Q-Learning (but of course, here we know the MDP we are in).

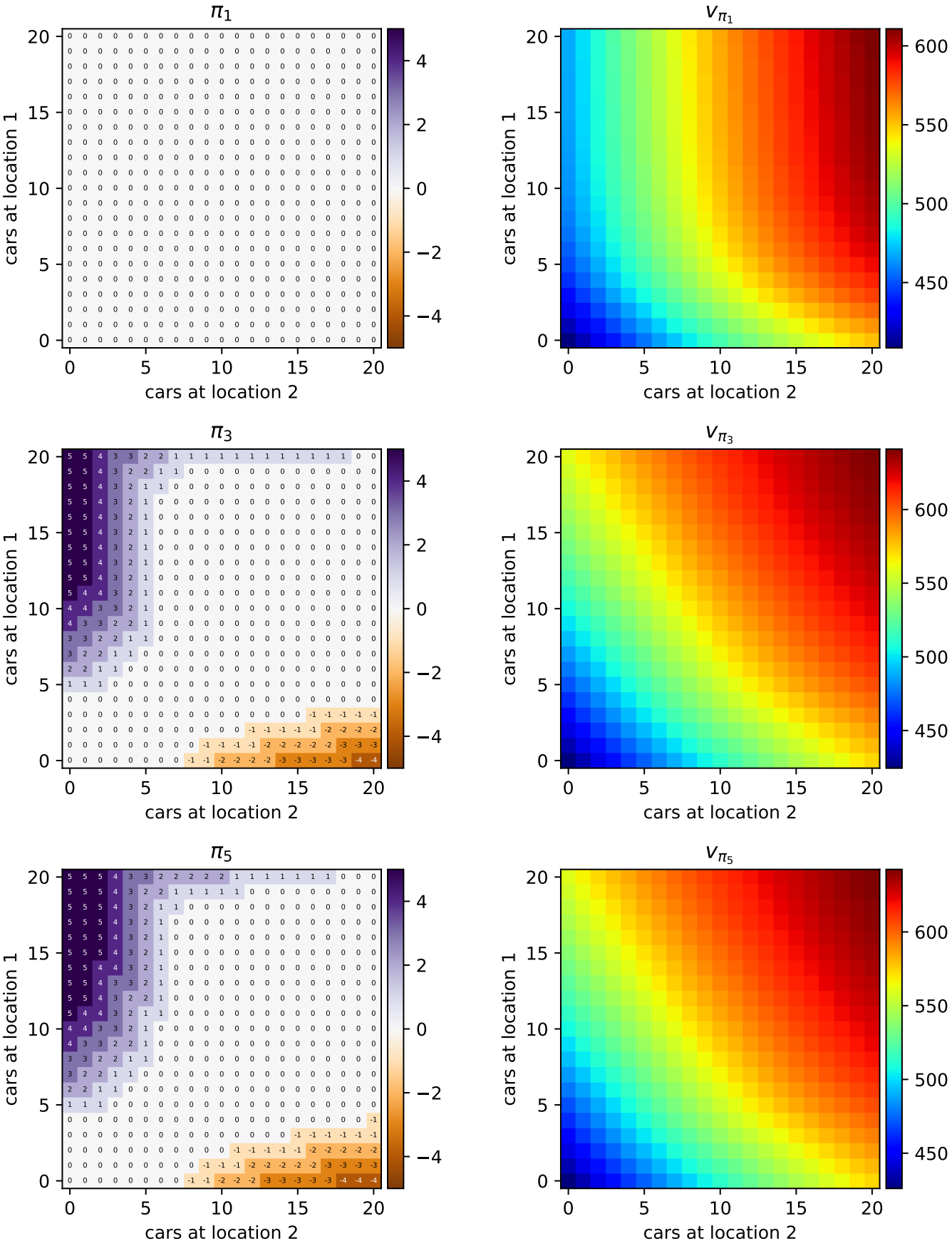

Figure 1: Replication of Fig. 4.2

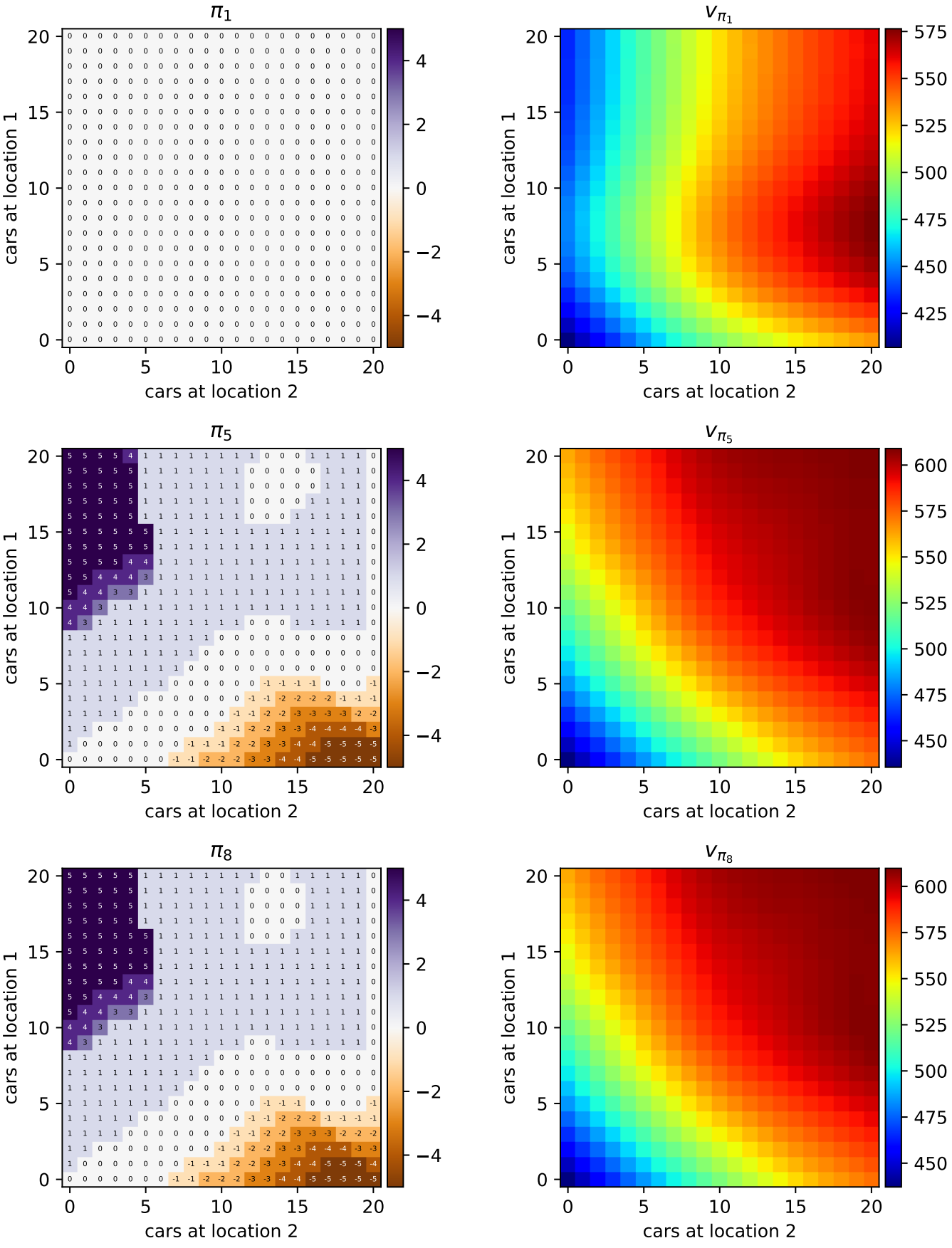

Figure 2: Modified Problem Solution

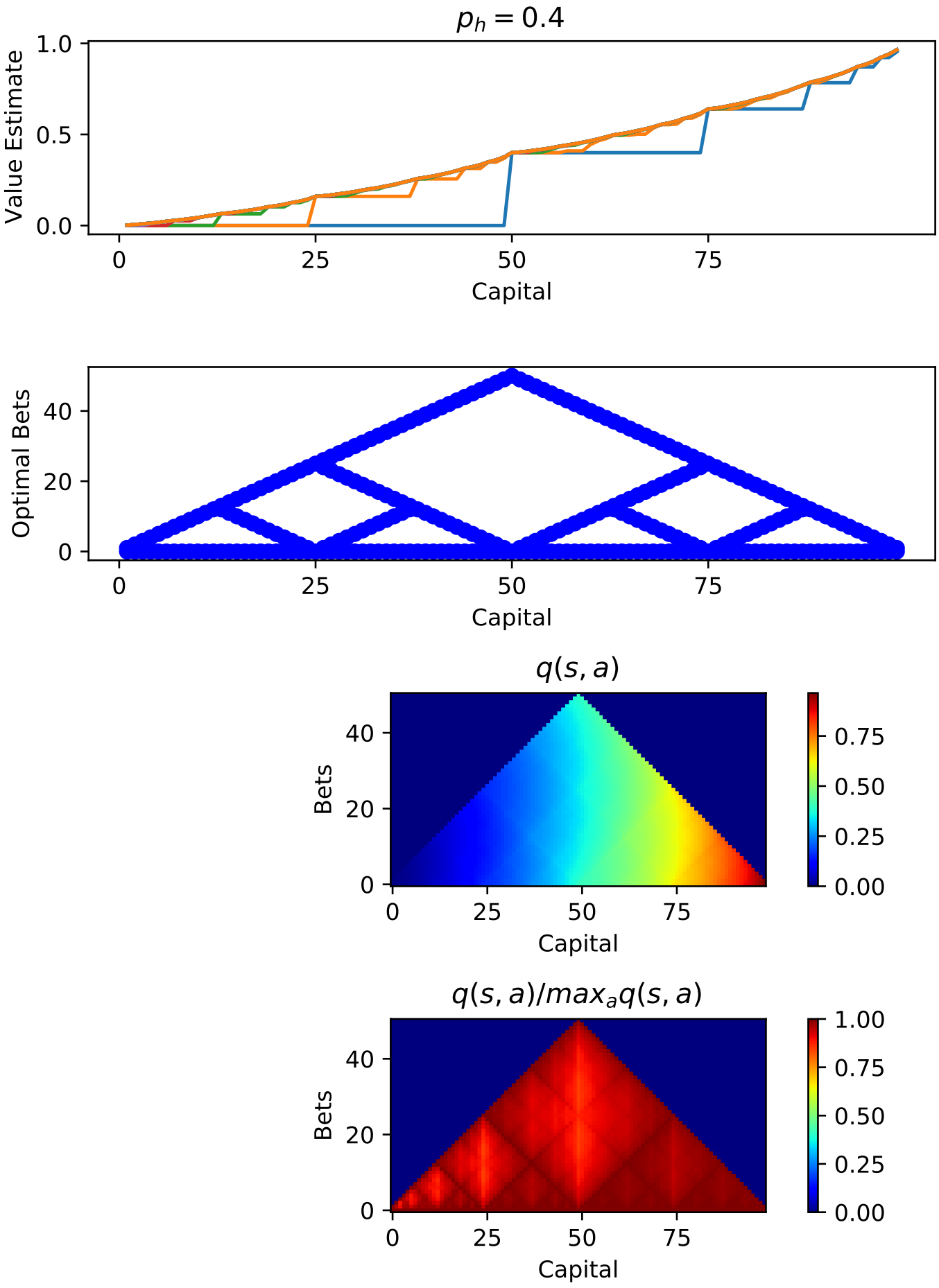

Figure 3: Reproduction of Gamblers Problem

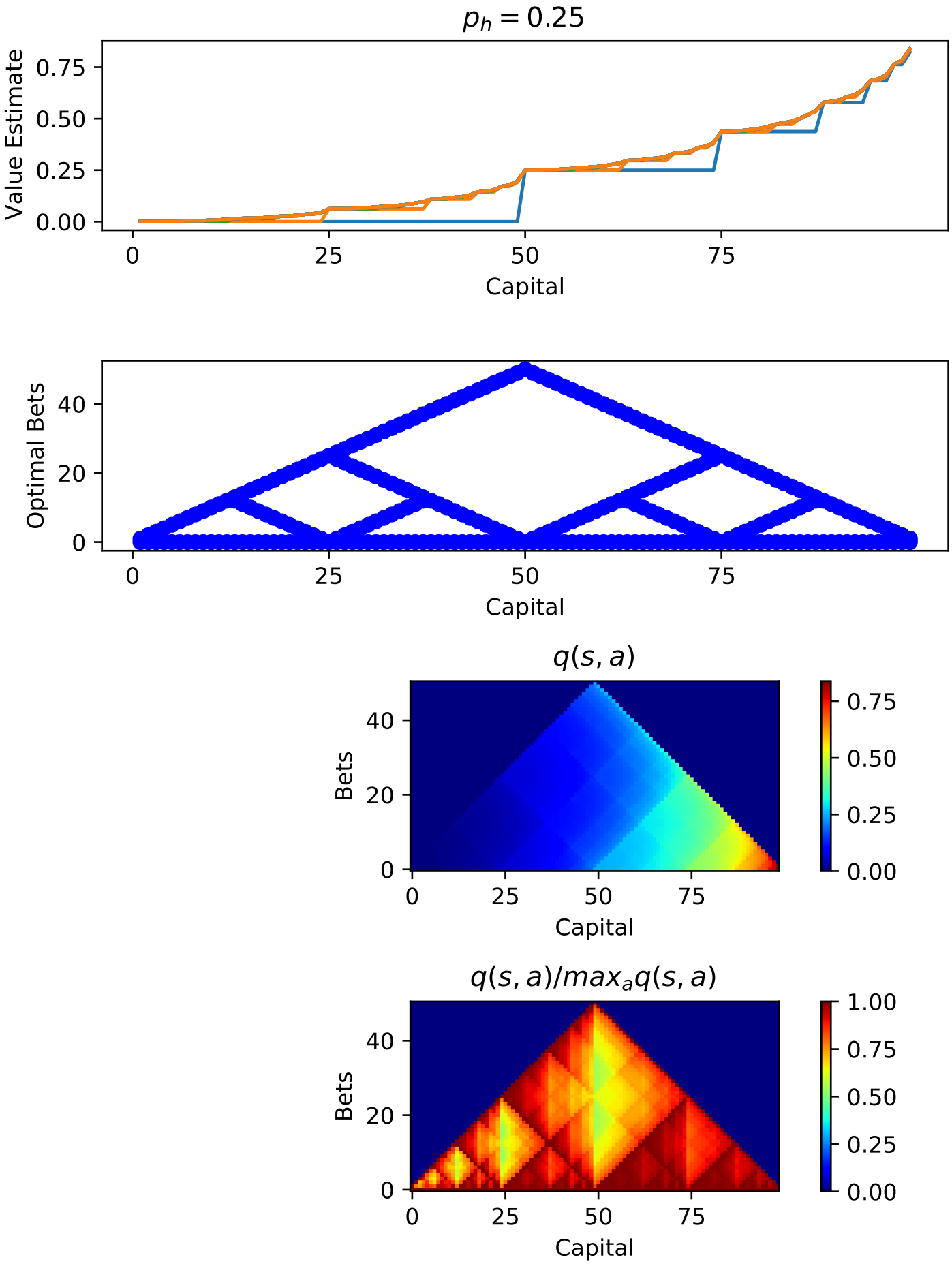

Figure 4: less chance of winning

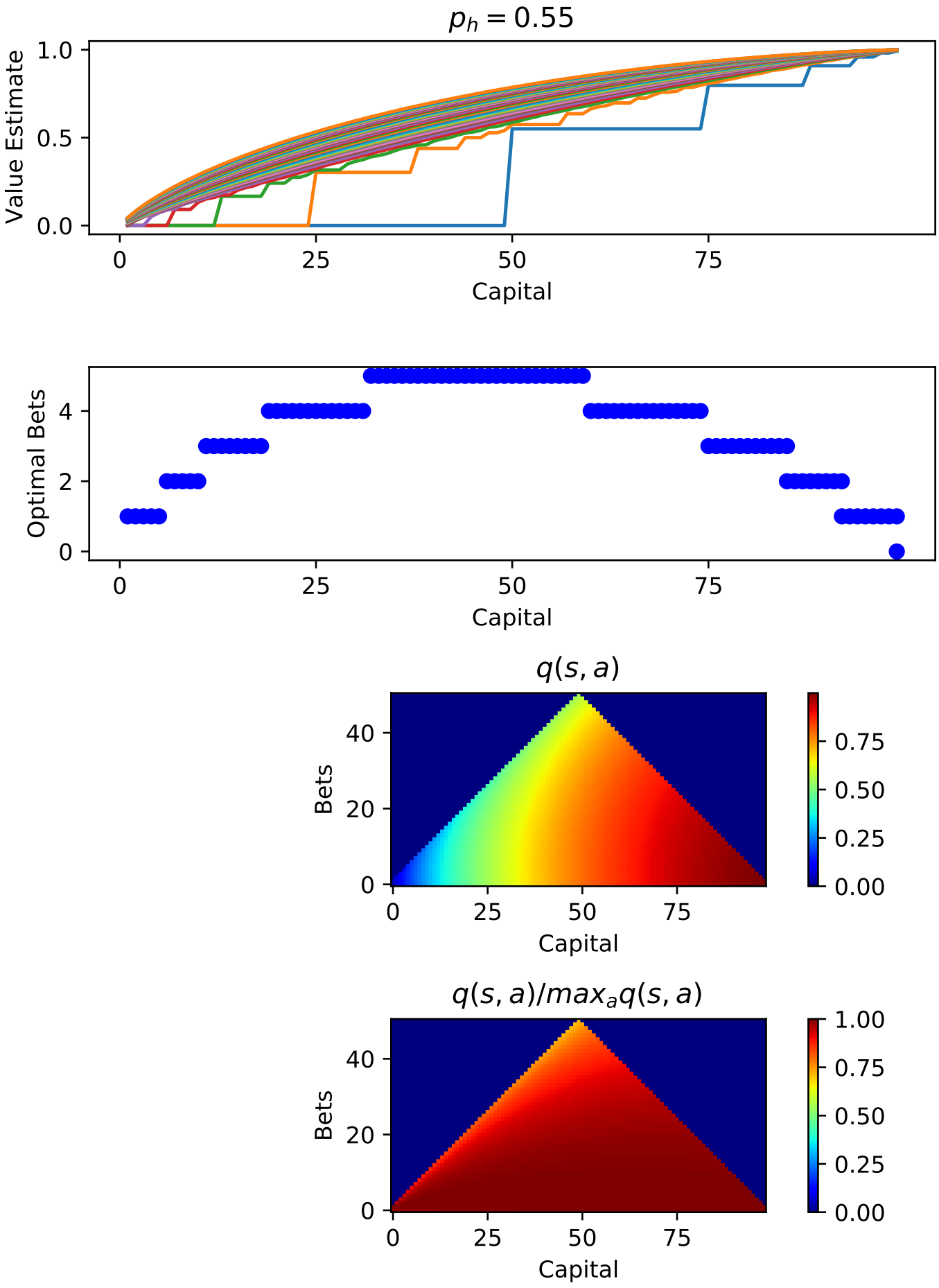

Figure 5: better chance of winning

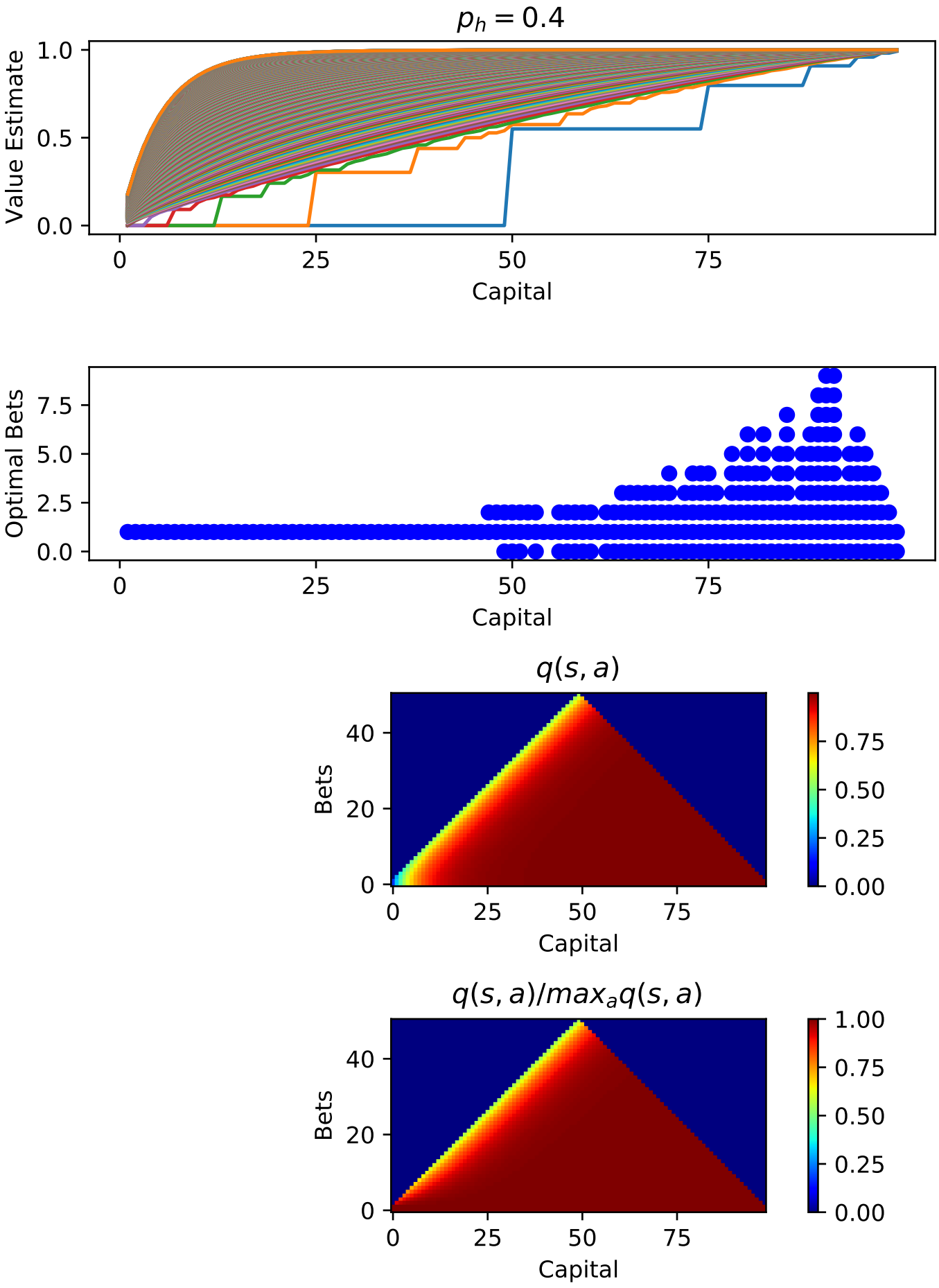

Figure 6: better chance of winning, 512 steps of value iteration

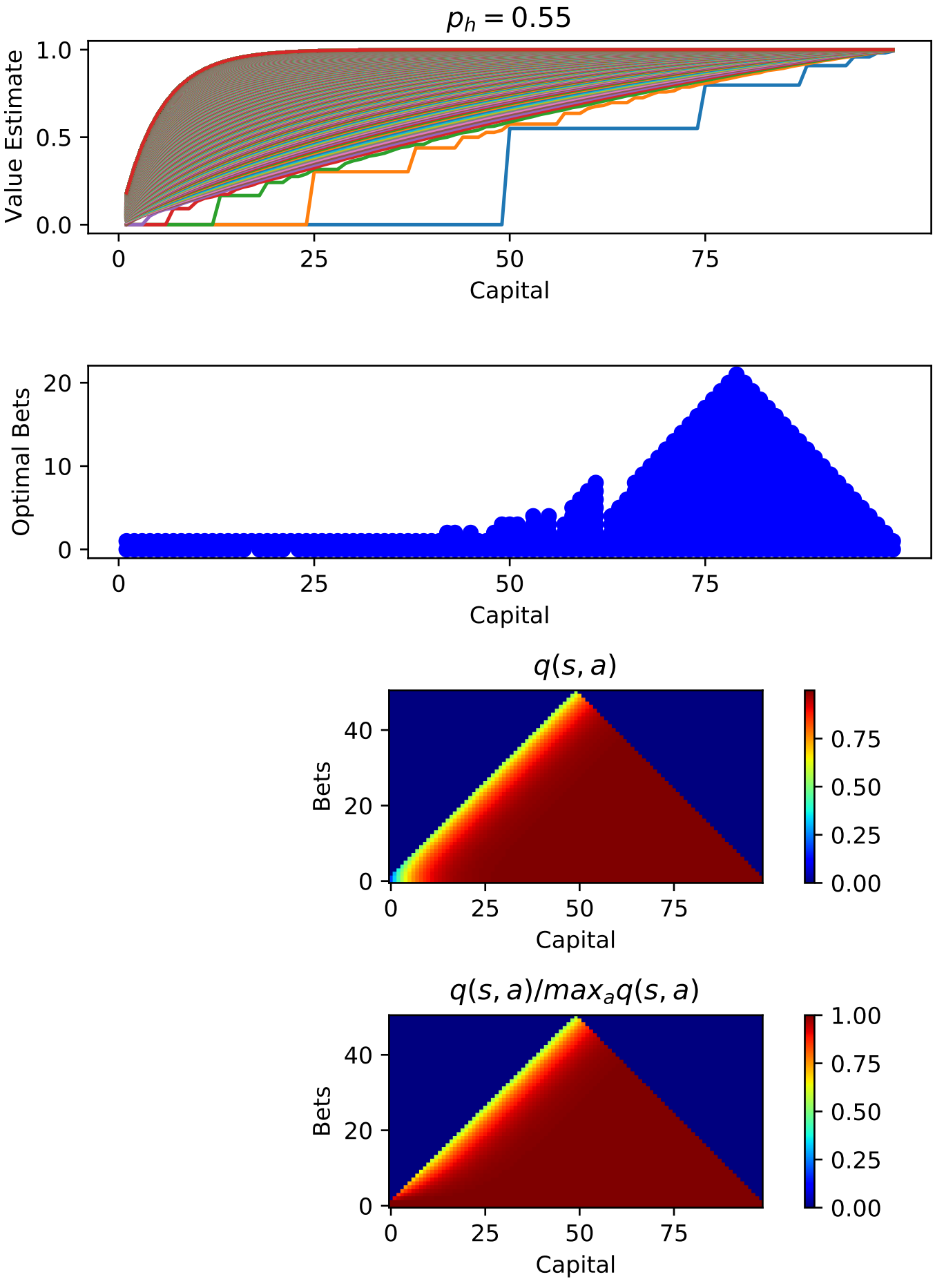

Figure 7: better chance of winning, 1024 steps of value iteration